DETAILED ACTION

1. Claims 1-20 are pending.

2. A request for continued examination under 37 CFR 1.114, including the fee set forth in 37 CFR 1.17(e), was filed in this application after final rejection. Since this application is eligible for continued examination under 37 CFR 1.114, and the fee set forth in 37 CFR 1.17(e) has been timely paid, the finality of the previous Office action has been withdrawn pursuant to 37 CFR 1.114. Applicant's submission filed on 8/10/2009 has been entered.

3. The office acknowledges the following papers:

Claims and arguments filed on 8/10/2009,

Power of Attorney filed on 8/13/2009.

New Claim Rejections - 35 USC § 103

4. The following is a quotation of 35 U.S.C. §103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

5. Claims 1-5 and 12-20 are rejected under 35 U.S.C. §103(a) as being unpatentable over Ramesh et al. (U.S. 6,651,159), in view of Tran et al. (U.S. 6,205,541).

6. As per claim 1:

Ramesh disclosed a method for managing type information for operands, the method comprising:

accomplishing the following results through execution of a single register instruction in a register of a processor (Ramesh: Figure 3 element 30, column 3 lines 25-49)(A single pop or push instruction uses a single register to add or delete data from the stack in the processor.):

adding an operand tag to a tag stack (Ramesh: Figure 4 element 33, column 3 lines 32-37 and column 4 lines 57-64)(When a push operation occurs, a tag for the operand is added to the pseudo-tag register.); and

updating a stack pointer for the tag stack by movement of the stack pointer to recognize the addition of the operand tag to the tag stack (Ramesh: Figure 5B, column 5 lines 55-63 continued to column 6 lines 1-6)(The top of stack pointer is incremented/decremented when data is added or deleted to the stack. It's obvious to one of ordinary skill in the art that a single instruction performs both the updating of the stack pointers and adding/deleting data to/from the stack and pseudo-tag register.), wherein a bit position of the stack pointer is to indicate a depth of the tag stack (Ramesh: Figure 5B, column 5 lines 55-63 continued to column 6 lines 1-6)(It's obvious to one of ordinary skill in the art that since the stack pointer is incremented/decremented for each stack pop/push operation, the stack pointer's position indicates the number of stack entries.).

Ramesh failed to teach the stack pointer stored in the tag stack and implicitly encoded.

However, Tran disclosed the stack pointer stored in the tag stack and implicitly encoded (Tran: Figure 5 element pointer0, column 15 lines 57-64)(Ramesh: Figure 5B, column 5 lines 55-63 continued to column 6 lines 1-6)(The combination keeps the stack pointer at a fixed position to implicitly encode and indicate the top of the stack without a stack pointer register.).

The advantage of using an implicit stack pointer is that chip space can be reduced by eliminating the stack pointer register that is otherwise required to store a stack pointer. One of ordinary skill in the art would have been motivated by this advantage to implement the implicit stack pointer of Tran in Ramesh. Thus, it would have been obvious to one of ordinary skill in the art at the time of the invention to implement an implicit stack pointer in Ramesh for the advantage of eliminating a stack pointer register to save chip space.

7. As per claim 2:

Ramesh and Tran disclosed a method according to claim 1, wherein the single register instruction comprises a shift instruction (Ramesh: Figure 5B, column 5 lines 55-63 continued to column 6 lines 1-6)(Data is shifted into the pseudo-tag register.).

8. As per claim 3:

Ramesh and Tran disclosed a method according to claim 2, wherein the shift instruction comprises a rotate instruction (Ramesh: Figure 5B, column 5 lines 55-63 continued to column 6 lines 1-6)(Data is shifted into the pseudo-tag register. A shift and rotate operation are one in the same when the rotate operation doesn't wrap around, which isn't claimed.).

9. As per claim 4:

Ramesh and Tran disclosed a method according to claim 1, further comprising:

accomplishing the following results through execution of one register instruction (Ramesh: Figure 3 element 30, column 3 lines 25-49)(A single pop or push instruction uses a single register to add or delete data from the stack in the processor.):

removing an operand tag from the tag stack (Ramesh: Figure 4 element 33, column 3 lines 38-49 and column 4 lines 57-64)(When a pop operation occurs, a tag for the operand is deleted from the pseudo-tag register.); and

updating the stack pointer for the tag stack to recognize the removal of the operand tag from the tag stack (Ramesh: Figure 5B, column 5 lines 55-63 continued to column 6 lines 1-6)(The top of stack pointer is updated when data is added or deleted to the stack. It's obvious to one of ordinary skill in the art that a single instruction performs both the updating of the stack pointers and adding/deleting data to/from the stack and pseudo-tag register.).

10. As per claim 5:

Ramesh and Tran disclosed a method according to claim 4, wherein the one register instruction comprises a shift right instruction (Ramesh: Figure 5B, column 5 lines 55-63 continued to column 6 lines 1-6)(Data is right-shifted into the pseudo-tag register when a pop instruction occurs.).

11. As per claim 12:

Ramesh disclosed a processing system with control logic for managing type information for operands, the processing system comprising:

a processor (Ramesh: Figure 1, column 2 lines 40-42);

a machine-accessible storage medium responsive to the processor (Ramesh: Figure 1)(The instruction executed by Ramesh are inherently stored within a memory within the processor of Ramesh.); and

instructions in the machine-accessible storage medium, the instructions to implement at least part of a virtual machine when executed by a processing system (Ramesh: Figure 1, column 2 lines 40-42)(The advantage of Java is that it's platform independent and is densely coded for compact programs to save memory space and power consumption. Official notice is given that processors can execute Java instructions. Thus, it's obvious to one of ordinary skill in the art to allow for the processor of Ramesh to execute Java instructions by implementing a virtual machine.), the virtual machine to accomplishing the following results through execution of a single register instruction (Ramesh: Figure 3 element 30, column 3 lines 25-49)(A single pop or push instruction uses a single register to add or delete data from the stack in the processor.):

adding an operand tag to a first order position of a tag stack to indicate a type of operand added to an operand stack (Ramesh: Figure 4 element 33, column 3 lines 32-37 and column 4 lines 57-64)(When a push operation occurs, a tag for the operand is added to the pseudo-tag register. The tag implicitly indicates that the operand is of a type since the operand inherently is referenced later or not referenced later.), wherein the operand tag is of a first value to indicate a reference type and of a second value to indicate a non-reference type (Ramesh: Figure 4 element 33, column 3 lines 32-37 and

column 4 lines 57-64)(When a push operation occurs, a tag for the operand is added to the pseudo-tag register. The tag implicitly indicates that the operand is of a type since the operand inherently is referenced later or not referenced later.); and

shifting a previous value stored in the first order position and all other order positions of the tag stack in a first direction (Ramesh: Figure 5B, column 5 lines 55-63 continued to column 6 lines 1-6)(Previous values are left-shifted for a push operation.).

updating a stack pointer for the tag stack by movement of the stack pointer to recognize the addition of the operand tag to the tag stack (Ramesh: Figure 5B, column 5 lines 55-63 continued to column 6 lines 1-6)(The top of stack pointer is incremented/decremented when data is added or deleted to the stack. It's obvious to one of ordinary skill in the art that a single instruction performs both the updating of the stack pointers and adding/deleting data to/from the stack and pseudo-tag register.), wherein a bit order position of the stack pointer is to indicate a depth of the tag stack (Ramesh: Figure 5B, column 5 lines 55-63 continued to column 6 lines 1-6)(It's obvious to one of ordinary skill in the art that since the stack pointer is incremented/decremented for each stack pop/push operation, the stack pointer's position indicates the number of stack entries.).

Ramesh failed to teach the stack pointer stored in the tag stack and implicitly encoded.

However, Tran disclosed the stack pointer stored in the tag stack and implicitly encoded (Tran: Figure 5 element pointer0, column 15 lines 57-64)(Ramesh: Figure 5B, column 5 lines 55-63 continued to column 6 lines 1-6)(The combination keeps the stack

pointer at a fixed position to implicitly encode and indicate the top of the stack without a stack pointer register.).

The advantage of using an implicit stack pointer is that chip space can be reduced by eliminating the stack pointer register that is otherwise required to store a stack pointer. One of ordinary skill in the art would have been motivated by this advantage to implement the implicit stack pointer of Tran in Ramesh. Thus, it would have been obvious to one of ordinary skill in the art at the time of the invention to implement an implicit stack pointer in Ramesh for the advantage of eliminating a stack pointer register to save chip space.

12. As per claim 13:

The additional limitation(s) of claim 13 basically recite the additional limitation(s) of claim 2. Therefore, claim 13 is rejected for the same reason(s) as claim 2.

13. As per claim 14:

The additional limitation(s) of claim 14 basically recite the additional limitation(s) of claim 3. Therefore, claim 14 is rejected for the same reason(s) as claim 3.

14. As per claim 15:

The additional limitation(s) of claim 15 basically recite the additional limitation(s) of claim 4. Therefore, claim 15 is rejected for the same reason(s) as claim 4.

15. As per claim 16:

Ramesh and Tran disclosed a processing system according to claim 12 wherein the processor supports a little-endian byte order (Official notice is given that processors store data in either little-endian or big-endian byte order. Thus, it's obvious to one of

ordinary skill in the art that the system of Sokolov stores data in little-endian byte order.).

16. As per claim 17:

Claim 17 essentially recites the same limitations of claim 12. Therefore, claim 17 is rejected for the same reasons as claim 12.

17. As per claim 18:

The additional limitation(s) of claim 18 basically recite the additional limitation(s) of claim 2. Therefore, claim 18 is rejected for the same reason(s) as claim 2.

18. As per claim 19:

The additional limitation(s) of claim 19 basically recite the additional limitation(s) of claim 3. Therefore, claim 19 is rejected for the same reason(s) as claim 3.

19. As per claim 20:

The additional limitation(s) of claim 20 basically recite the additional limitation(s) of claim 4. Therefore, claim 20 is rejected for the same reason(s) as claim 4.

20. Claims 6-11 are rejected under 35 U.S.C. §103(a) as being unpatentable over Ramesh et al. (U.S. 6,651,159), in view of Adl-Tabatabai et al. (U.S. 6,317,869).

21. As per claim 6:

Ramesh and Adl-Tabatabai disclosed a method for managing type information for operands, the method comprising:

shifting a bit value of 1 into a first significant bit of a register and shifting all bits of the register in a first direction, in conjunction with creation of a reference operand

(Ramesh: Figure 5B, column 4 lines 57-64 and column 5 lines 55-67 continued to column 6 lines 1-6)(Adl-Tabatabai: Figures 4b and 5b element 555, column 6 lines 39-56)(Ramesh disclosed a pseudo-tag register that shifts in 1 values when data is added to the stack register. The combination results in a 1 shifted into the pseudo-tag register when the value is a reference operand.); and

shifting a bit value of 0 into the first significant bit of the register and shifting all bits of the register in the first direction, in conjunction with creation of a non-reference operand (Ramesh: Figure 5B, column 4 lines 57-64 and column 5 lines 55-67 continued to column 6 lines 1-6)(Adl-Tabatabai: Figures 4b and 5b element 555, column 6 lines 39-56)(Ramesh disclosed a pseudo-tag register that shifts in 1 values when data is added to the stack register. The combination results in a 0 shifted into the pseudo-tag register when the value is a non-reference operand.).

The advantage of storing a 1 or 0 in the pseudo-tag register of Ramesh is that it can keep track of references to assist in garbage collection for executing a JAVA program. One of ordinary skill in the art would have been motivated by this advantage to implement reference tracking for JAVA programs in Ramesh. Thus, it would have been obvious to one of ordinary skill in the art at the time of the invention to implement reference tracking for JAVA programs in Ramesh for the advantage of assisting in garbage collection for JAVA programs.

22. As per claim 7:

Ramesh and Adl-Tabatabai disclosed a method according to claim 6, wherein:

the register serves as a tag stack register, the tag stack register to be used for storing a stack of operand tags (Ramesh: Figure 3a element 33, column 4 lines 57-64)(Adl-Tabatabai: Figure 4b, column 6 lines 8-16 and 25-28)(The combination uses the pseudo-tag register to store operand tags.), each operand tag to indicate whether a corresponding operand on an operand stack is to be treated as a reference operand or a non-reference operand (Ramesh: Figure 3a element 33, column 4 lines 57-64)(Adl-Tabatabai: Figure 4b, column 6 lines 8-16)(The combination uses the pseudo-tag register to store operand tags. The combination uses 1 or 0 bits indicate if an operand makes a reference to an object or not.); and

the method further comprises initializing the tag stack register by:

assigning a low order bit of the tag stack register to a value of 0 to implicitly encode a stack pointer (Ramesh: Figure 5b, column 5 lines 55-67 continued to column 6 lines 1-6)(The "ffree" operation can set the stack pointer as zero.), where a bit position of the stack pointer is to indicate a depth of the stack of operand tags (Ramesh: Figure 5B, column 5 lines 55-63 continued to column 6 lines 1-6)(It's obvious to one of ordinary skill in the art that since the stack pointer is incremented/decremented for each stack pop/push operation, the stack pointer's position indicates the number of stack entries.); and

assigning other bits of the tag stack register to a value of 1 (Ramesh: Figure 5b, column 5 lines 55-67 continued to column 6 lines 1-6)(The "ffree" operation can set the stack pointer as zero. Other bits are assigned to a "1" value.).

23. As per claim 8:

Ramesh and Adl-Tabatabai disclosed a method according to claim 6, further comprising:

using a shift left operation to shift a bit value into a low order bit of the register and shift a bit value of preceding order bits of the register into succeeding order bits of the register in response to an operand being added to an operand stack (Ramesh: Figure 5b, column 5 lines 55-67 continued to column 6 lines 1-6)(A push operation performs a left shift into the pseudo-tag register.).

24. As per claim 9:

Ramesh and Adl-Tabatabai disclosed a method according to claim 6, further comprising:

right shifting bit values in the register in conjunction with removal of an operand (Ramesh: Figure 5b, column 5 lines 55-67 continued to column 6 lines 1-6)(A pop operation performs a right shift into the pseudo-tag register.), the operand being one of the reference operand and the non-reference operand (Ramesh: Figure 5b, column 5 lines 55-67 continued to column 6 lines 1-6)(Adl-Tabatabai: Column 6 lines 49-57)(The combination results in the tag being removed being either referenced or non-referenced.).

25. As per claim 10:

Ramesh and Adl-Tabatabai disclosed a method according to claim 9, further comprising:

shifting the bit value of 1 into a high order bit of the register in conjunction with removal of the operand (Ramesh: Figure 5b, column 5 lines 55-67 continued to column

6 lines 1-6)(A pop operation performs a right shift into the pseudo-tag register. A zero instead of a one is shifted in. However, it's obvious to one of ordinary skill in the art that a one can be shifted in for the same result.).

26. As per claim 11:

Ramesh and Adl-Tabatabai disclosed a method according to claim 1 further comprising:

treating a highest order bit with the value of 0 in the tag stack register as a stack pointer (Ramesh: Figure 5b, column 5 lines 55-67 continued to column 6 lines 1-6)(The "ffree" operation can set the stack pointer as zero.); and

determining the depth of the stack of operand tags, based at least in part on a location of the stack pointer (Ramesh: Figure 5B, column 5 lines 55-63 continued to column 6 lines 1-6)(It's obvious to one of ordinary skill in the art that since the stack pointer is incremented/decremented for each stack pop/push operation, the stack pointer's position indicates the number of stack entries.).

Response to Arguments

27. The arguments presented by Applicant in the response, received on 8/10/2009 are partially considered persuasive.

28. Applicant argues "As to claim 1, Ramesh fails to teach that a stack pointer is updated by movement of the stack pointer, where the stack pointer is stored in a tag stack and is implicitly encoded. First, Ramesh fails to teach that a stack pointer be stored in a tag stack. Instead, Ramesh teaches that a top of stack (TOS) pointer is

separate from data stored in a register stack."

This argument is found to be persuasive for the following reason. The examiner agrees that Ramesh failed to teach "where the stack pointer is stored in a tag stack and is implicitly encoded." However, a new ground of rejection has been given due to the amendment.

29. Applicant argues "Still further regarding claim 12, Ramesh fails to anywhere teach or suggest the recited operand tag that indicates a type of operand (either reference or non-reference type) added to an operand stack. Still further, nothing in the reference anywhere teaches adding this operand tag to a first order position of a tag stack. In addition, the reference is silent with regard to any shifting, and certainly teaches nothing with regard to the recited shifting of a previous value stored in the positions of the tag stack in a first direction."

This argument isn't found to be persuasive for the following reason. The addition of a bit to the pseudo-tag register implicitly implies that an operand added to the register stack is either a reference type operand (i.e. used by other instructions) or a non-reference type operand (i.e. not used by other instructions). Figure 5b shows that the operand tags in the pseudo-tag register are shifted in response to push and pop operations, wherein the pseudo-tag register contains previous values.

30. Applicant argues "Thus each bit of the bit vector is dedicated to a given variable. In contrast, claim 6 recites that when a reference operand is created, a given bit value is shifted into a single location of a register, namely the first significant bit, regardless of what reference operand is created. Thus in claim 6 every time a reference operand is

created, a given bit value is shifted into the first significant bit of a register. In contrast, the reference does not shift a value into this location when almost all operands are created. Still further, the reference fails to teach that all bits of the register are shifted in a first direction in conjunction with creation of a reference operand."

This argument is found to be persuasive for the following reason. The examiner agrees that the reference failed to teach the newly claimed limitations. However, a new ground of rejection has been given due to the amendment.

Conclusion

The following is text cited from 37 CFR 1.111(c): In amending in reply to a rejection of claims in an application or patent under reexamination, the applicant or patent owner must clearly point out the patentable novelty which he or she thinks the claims present in view of the state of the art disclosed by the references cited or the objections made. The applicant or patent owner must also show how the amendments avoid such references or objections.

Any inquiry concerning this communication or earlier communications from the examiner should be directed to Jacob Petranek whose telephone number is 571-272-5988. The examiner can normally be reached on M-F 8:00-4:30.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Eddie Chan can be reached on (571) 272-4162. The fax phone number for the organization where this application or proceeding is assigned is 703-872-9306.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

/Jacob Petranek/
Examiner, Art Unit 2183